Approximation algorithms for precedence-constrained scheduling problems on parallel machines that run at different speeds

Fabián A. Chudak* David B. Shmoys†

Abstract

We present new approximation algorithms for the problem of scheduling precedence-constrained jobs on parallel machines that are uniformly related. That is, there are n jobs and m machines; each job j requires p_j units of processing, and is to be processed on one machine without interruption; if it is assigned to machine i, which runs at a given speed s_i, it takes p_j/s_i time units. There also is a partial order $\prec$ on the jobs, where $j \prec k$ implies that job k may not start processing until job j has been completed. We shall consider two objective functions: $C_{\max} = \max_j C_j$, where C_j denotes the completion time of job j, and $\sum_j w_j C_j$, where w_j is a weight that is given for each job j.

For the first objective, the best previously known result is an $O(\sqrt{m})$-approximation algorithm, which was shown by Jaffe more than 15 years ago. We shall give an $O(\log m)$-approximation algorithm. We shall also show how to extend this result to obtain an $O(\log m)$-approximation algorithm for the second objective, albeit with a somewhat larger constant. These results also extend to settings in which each job j has a release date r_j before which the job may not begin processing. In addition, we obtain stronger performance guarantees if there are a limited number of distinct speeds. Our results are based on a new linear programming-based technique for estimating the speed at which each job should be run, and a variant of the list scheduling algorithm of Graham that can exploit this additional information.

1 Introduction

The study of performance guarantees for approximation algorithms can be traced back to the work of Graham (1966), who studied the following scheduling problem. There are n jobs to be scheduled on m identical parallel machines, where each job j, $j = 1, \ldots, n$, requires p_j uninterrupted units of processing on one of the machines, and each machine can process at most one job at a time; furthermore, there is a partial order $\prec$ on the jobs that specifies precedence constraints, where $j \prec k$ means that the processing of job j must be completed by the time job k is started. The objective is to minimize the length of the schedule, that is, the time by which all jobs have been completed. Graham showed that a simple list scheduling rule finds a schedule of length within a factor of 2 of optimum, and no better constant guarantee is known today. We shall say that an algorithm is a ρ-approximation algorithm if it is guaranteed to deliver a solution of objective function value within a factor of ρ of optimal in polynomial time.

Our work concerns a natural generalization of this problem, in which each machine i runs at a specified speed s_i, $i = 1, \ldots, m$, and so if job j is processed by machine i, it takes p_j/s_i time units

*`chudak@cs.cornell.edu`. School of Operations Research & Industrial Engineering, Cornell University, Ithaca, NY 14853. Research partially supported by NSF grants CCR-9307391 and DMI-9157199.

†`shmoys@cs.cornell.edu`. School of Operations Research & Industrial Engineering and Department of Computer Science, Cornell University, Ithaca, NY 14853. Research partially supported by NSF grants CCR-9307391 & DMS-9505155 and ONR grant N00014-96-1-0050O.

to be completed, $i = 1, \ldots, m$, $j = 1, \ldots, n$; such parallel machines are called *uniformly related.* Liu & Liu (1974) showed that one could still analyze the list scheduling approach in this setting, but they proved a performance guarantee that depends on the particular speeds of the machines and, even for a fixed number of machines, this guarantee can be arbitrarily large. Jaffe (1980) refined this analysis, and showed that if one restricts attention to those machines whose speeds are within a factor of $\sqrt{m}$ of the fastest machine, and applies list scheduling using only those machines, then the resulting algorithm is an $O(\sqrt{m})$-approximation algorithm. Prior to our work, this was the best known performance guarantee for the problem of scheduling on uniformly related parallel machine subject to precedence constraints. We shall provide an $O(\log m)$-approximation algorithm for this problem. Furthermore, we shall provide a much stronger performance guarantee for instances in which there are a limited number of distinct machine speeds.

If all of the jobs are independent, or in other words, there are no precedence constraints, then stronger performance guarantees are known. Gonzalez, Ibarra, & Sahni (1977) gave a simple 2-approximation algorithm; Hochbaum & Shmoys (1988) gave a polynomial approximation scheme, or in other words, provided a ρ-approximation algorithm for each $\rho > 1$. A result of Lenstra & Rinnooy Kan (1978) implies that if there are precedence constraints, then, for each $\rho < 4/3$, no ρ-approximation algorithm exists unless $\mathcal{P} = \mathcal{NP}$. The performance guarantees that have been obtained for machines that run at different speeds have typically matched those obtained for the special case in which the machines are identical. This analogy would be complete if one could show that there exists a constant approximation algorithm (or perhaps even a 2-approximation algorithm) for the precedence-constrained problem on uniformly related parallel machines. By improving the best known performance guarantee in this case from $O(\sqrt{m})$ to $O(\log m)$, we are taking a significant step towards this goal.

Throughout this paper, we shall use the following notation, which follows the survey article of Graham, Lawler, Lenstra, & Rinnooy Kan (1979). For a given schedule, C_j shall denote the completion time of job j, $j = 1, \ldots, n$. Thus, the length of the schedule is $\max_{j=1,\ldots,n} C_j$, and we shall denote this by $C_{\max}$. We shall also be interested in minimizing the weighted sum of job completion times; that is, each job j has a given weight w_j, $j = 1, \ldots, n$, and the objective is to minimize $\sum_{j=1}^{n} w_j C_j$. We shall also give an $O(\log m)$-approximation algorithm for this objective, which improves upon an $O(\sqrt{m})$ performance guarantee proved by Schulz (1996).

We shall also adopt the $\alpha|\beta|\gamma$ notation of Graham, Lawler, Lenstra, & Rinnooy Kan (1979) to specify scheduling problems: we shall only be interested in the cases in which α is either P or Q, denoting identical parallel machines or uniformly related parallel machines, respectively; β is a (possibly empty) subset of $\{r_j, prec, pmtn\}$, where r_j indicates that each job j has a specified release date before which it may not start processing, *prec* indicates that there are precedence constraints, and *pmtn* indicates that preemption is allowed, (that is, the processing of a job may be interrupted and continued later, possibly in a different machine); γ indicates the objective function, which will be either $C_{\max}$ or $\sum w_j C_j$. For example, the main results of this paper are $O(\log m)$-approximation algorithms for the problems $Q|r_j, prec|C_{\max}$ and $Q|r_j, prec|\sum w_j C_j$.

The list scheduling algorithm of Graham can be described as follows. The jobs are listed in some order, for example, $1, 2, \ldots, n$. The schedule is constructed "in time": whenever a job completes, each idle machine selects the first as yet unscheduled job on the list for which all of its predecessors have been completed; if no such job exists, then the machine remains idle until the next job has been completed.

Our algorithm is based on the following simple ideas. We consider only the machines whose speeds are within a factor of m of the fastest machine, and partition the machines into $\log m$ groups so that, within a group, all machines have roughly the same speed (for example, within a factor of 2 of each other). We then assign each job to one of these groups; each job will be processed by one of the machines in its assigned group. The schedule is then constructed by the following slight variant of the list scheduling algorithm: each machine may only select a job for processing if this job has been assigned to its group. Of course, the performance of the algorithm depends critically on the way in which the job assignments are done. Our assignment algorithm is based on solving a linear programming relaxation of the problem, in order to determine an estimate of the speed at which each job should be done; then the final assignment ensures that the job is processed at least at that speed, while also assuring that the loads on the various groups are sufficiently well balanced.

In several respects, this algorithm is similar to an on-line $O(\log m)$-approximation algorithm of Shmoys, Wein, & Williamson (1995) for $Q||C_{\max}$. For their algorithm, each job assignment was based simply on whether the job could finish on a machine in that group by a currently estimated deadline. Our algorithm was also motivated by recent results of Hall, Schulz, Shmoys, & Wein (1996) and Chakrabarti, Phillips, Schulz, Shmoys, Stein & Wein (1996) that employ linear programming in the design of list scheduling algorithms for minimizing $\sum w_j C_j$. In these algorithms, the optimal solution to a linear programming relaxation was used to order the jobs in the list. We will also take advantage of these ideas in our algorithm for the $\sum w_j C_j$ objective.

Finally, we consider the generalizations of these problems in which each job has a specified release date r_j, before which it is not allowed to begin processing. For the $C_{\max}$ objective, this is not a substantially more general problem. Shmoys, Wein, & Williamson (1995) have given a broadly applicable technique that takes a ρ-approximation algorithm for a scheduling problem $\alpha|\beta|C_{\max}$ without release dates, and converts it into a 2ρ-approximation algorithm for the generalization with release dates; in fact, the new algorithm is *on-line*, in the sense that for each point in time t, the algorithm schedules up to time t without knowledge of those jobs released at time t or later. This technique is quite simple: the jobs are scheduled in batches, where the jobs in the next batch are simply those jobs that were released while the jobs in the previous batch were being processed. Hence, an immediate corollary of our work is an $O(\log m)$-approximation algorithm for the generalization with release dates. For the $\sum w_j C_j$ objective, we can incorporate the release-date constraints into our linear programming formulation, and obtain a relatively direct extension in this way.

2 A speed-based list scheduling algorithm

In this section, we will introduce our variant of the list-scheduling algorithm that is the basis for our approximation algorithms for scheduling on uniformly related machines. The proof of the performance guarantee of the list scheduling algorithm for identical parallel machines is quite simple. Graham (1966) observed that any schedule produced by this algorithm has a chain of jobs $j_1 \prec j_2 \prec \cdots \prec j_r$ such that a machine is idle only when some job in the chain is being processed. The total processing requirement of any chain is a lower bound on $C^*_{\max}$, the length of the optimal schedule. Similarly, the total length of time intervals in which all machines are busy is also a lower bound on $C^*_{\max}$. Hence, the total length of the schedule is at most $2C^*_{\max}$.

If one considers the straightforward extension of this analysis to uniformly related parallel machines, then the main difficulty is that the first lower bound no longer holds. The time required to process a particular chain of jobs can only be lower bounded by the time required on the fastest machine, and the list scheduling algorithm might not have scheduled these jobs on the fastest machine. However, if we know in advance at which speed to process each job, then the length of time spent processing a chain can provide a lower bound once again. Unfortunately, if we are restricted to process jobs at particular speeds, then we can no longer ensure that *all* machines are busy. However, we can ensure something slightly weaker, that at least all machines of one particular speed are busy, and this is the central idea behind our algorithm.

Assume that there are m_k machines of speed $\overline{s}_k$, $k = 1, \ldots, K$, where $\overline{s}_1 > \overline{s}_2 > \cdots > \overline{s}_K$; furthermore, for each job j, $j = 1, \ldots, n$, let $k(j)$ index the speed at which job j is to be processed. For this assignment, we can compute the total load D_k assigned to machines of speed $\overline{s}_k$; that is,

$$D_k = \frac{1}{m_k} \sum_{j:k(j)=k} \frac{p_j}{\overline{s}_k}.$$

We can also bound the length of each chain of jobs: let C be the maximum over all chains $\mathcal{C}$ of

$$\sum_{j \in \mathcal{C}} \frac{p_j}{\overline{s}_{k(j)}}.$$

Consider the following speed-based list scheduling algorithm to schedule the jobs listed in the order $1, 2, \ldots, n$. The schedule is constructed "in time"; whenever a job completes, we attempt to start a job on each machine that is not currently processing a job. We consider the idle machines in some given order. For a machine of speed $\overline{s}_k$, we select the first job j on the list that has not already been scheduled for which $k(j) = k$ and all of its predecessors have been completed; if no such job exists, then this machine remains idle until the next job has been completed.

The speed-based list scheduling algorithm clearly produces a feasible schedule. We will next show how to bound the length of this schedule.

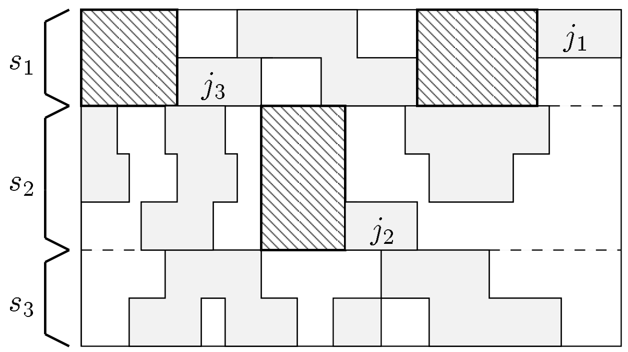

Figure 1: Scheduling by the speed-based algorithm.

Theorem 2.1 *For any job assignment $k(j)$, $j = 1, \ldots, n$, the speed-based list scheduling algorithm produces a schedule of length*

$$C_{\max} \leq C + \sum_{k=1}^{K} D_k.$$

Proof: The proof of this theorem is a generalization of the analysis of Graham (1966). Consider the schedule produced by our algorithm, as depicted in Figure 1. We will partition the time from 0 to $C_{\max}$ into $K + 1$ parts, $\mathcal{T}_k$, $k = 0, \ldots, K$, where each $\mathcal{T}_k$ consists of the union of disjoint time intervals. Consider a job j_1 that completes at time $C_{\max}$. We shall construct a chain that ends with j_1 as follows: for $t = 1, 2, \ldots,$ iteratively define j_{t+1} as a predecessor of j_t that completes last in the schedule; if j_t does not have any predecessors, then the chain $\mathcal{C}$ is complete. Let $\mathcal{T}_0$ denote the periods of time during which some job in $\mathcal{C}$ is being processed. Consider the complementary time periods, which consist of a collection of disjoint time intervals, each of which ends at a time that some job in $\mathcal{C}$ begins processing. Consider the interval that ends when job $j \in \mathcal{C}$ begins processing and assign this interval to $\mathcal{T}_{k(j)}$. We have now partitioned the time from 0 to $C_{\max}$ into the sets $\mathcal{T}_k$, $k = 0, \ldots, K$.

First of all it is clear that the total length of $\mathcal{T}_0$ is at most C, since the algorithm assigns each job j to be processed on a machine of speed $\bar{s}_{k(j)}$, $j = 1, \ldots, n$. Next, we argue that, during each time interval included in $\mathcal{T}_k$, no machine of speed $\bar{s}_k$ is idle, $k = 1, \ldots, K$. The reason for this is simple. Consider one such interval I assigned to $\mathcal{T}_k$, for some $k = 1, \ldots, K$, and the job $j \in \mathcal{C}$ that starts at its endpoint. Our construction ensures that $k(j) = k$ and that all predecessors of j have completed by the start of I (since the immediate predecessor j' of j in $\mathcal{C}$ completes at the start of I, and j' was selected as the predecessor of j that is the last to complete). Consequently, if a machine of speed $\bar{s}_k$ were idle in I, then job j would have been scheduled earlier by the algorithm. Hence, all machines of speed $\bar{s}_k$ are busy at each time in $\mathcal{T}_k$. If the total length of the intervals in $\mathcal{T}_k$ is t_k, then there must be $\bar{s}_k m_k t_k$ units of processing done on these machines during this time. However, there are only $\sum_{j:k(j)=k} p_j$ units of processing to be performed on machines of speed $\bar{s}_k$ throughout the entire schedule. Hence,

$$t_k \le \frac{1}{m_k \bar{s}_k} \sum_{j:k(j)=k} p_j = D_k, \qquad k = 1, \ldots, K.$$

Therefore, we have shown that the length of the entire schedule is at most $C + \sum_{k=1}^{K} D_k$. ■

3 An $O(\log m)$-approximation algorithm for $Q|r_j, prec|C_{\max}$

In this section, we will show how to assign jobs to machine speeds to yield a $(K + 2\sqrt{K} + 1)$-approximation algorithm for inputs in which there are K distinct machine speeds. We then show how this algorithm can be modified to obtain an $O(\log m)$-approximation algorithm in general. Furthermore, by a general result of Shmoys, Wein, & Williamson (1995), this implies that there also is an $O(\log m)$-approximation algorithm for the more general setting in which each job j has a specified release date r_j before which it may not begin processing.

As before, suppose that there are m_k machines of speed $\bar{s}_k$, where $\bar{s}_1 > \bar{s}_2 > \cdots > \bar{s}_K$. Then, in any schedule of length $C_{\max}$, the kth group of machines can perform at most $m_k \bar{s}_k C_{\max}$ units of processing. If we schedule jobs on only the group of machines for which this capacity is largest, then we are increasing the total load on this group by at most a factor of K. Of course, if this group of machines is also the slowest, then we might greatly increase the length of time needed to process any chain. Hence, we will first compute a good estimate for the length of time that is "reasonable" to spend processing each job, and then assign the job to be processed on the greatest capacity group of the machines that run at least that fast. The estimates will be computed by solving a relaxation of $Q|prec|C_{\max}$.

We would like to assign jobs to machine speeds in such a way that the upper bound $C + \sum_{k=1}^{K} D_k$ is not too large. We will introduce a relaxation of the problem of computing an assignment that corresponds to a schedule of length D. We shall use 0-1 variables x_{kj} to indicate whether or not job j is assigned to run at speed $\bar{s}_k$. Since each job is to be assigned to some machine speed,

$$\sum_{k=1}^{K} x_{kj} = 1, \qquad j = 1, \ldots, n. \tag{1}$$

The total processing requirement assigned to machines of speed $\bar{s}_k$ does not exceed the capacity that can be processed by these machines by time D:

$$\frac{1}{m_k \overline{s}_k} \sum_{j=1}^{n} p_j x_{kj} \leq D, \qquad k = 1, \ldots, K. \tag{2}$$

To bound the length of each chain, we introduce variables C_j, $j = 1, \ldots, n$, which are *not* constrained to be integer, and represent completion times. Clearly, each job cannot be completed earlier than the time required to process it on its assigned machine:

$$\sum_{k=1}^{K} \frac{p_j}{\overline{s}_k} x_{kj} \leq C_j, \qquad j = 1, \ldots, n. \tag{3}$$

Furthermore, for each precedence relation $j' \prec j$, the difference $C_j - C_{j'}$ must be at least the time spent processing j:

$$\sum_{k=1}^{K} \frac{p_j}{\overline{s}_k} x_{kj} \leq C_j - C_{j'}, \qquad \text{if } j' \prec j. \tag{4}$$

Of course, if the schedule is of length D, each job must complete by then:

$$C_j \leq D, \qquad j = 1, \ldots, n. \tag{5}$$

Finally, the assignment variables are constrained to be 0-1:

$$x_{kj} \in \{0, 1\}, \qquad k = 1, \ldots, K, \; j = 1, \ldots, n. \tag{6}$$

Hence, a mixed-integer programming relaxation of $Q|prec|C_{\max}$ is as follows: minimize D subject to (1)-(6); that is, the optimal solution to this mixed-integer program gives a lower bound on the length of the optimal schedule, $C^*_{\max}$.

For any assignment that corresponds to a feasible solution for this mixed-integer program of objective function value D, the speed-based list scheduling algorithm will find a schedule of length $(K+1)D$: the constraints (4) & (5) imply that the length of any chain is at most D, and the constraints (2) imply that for each $k = 1, \ldots, K$, the total load D_k assigned to the machines of speed $\overline{s}_k$ is at most D. In fact, Theorem 2.1 states something much stronger: in order to obtain this bound we do not need an integer solution that satisfies (2), but rather just satisfies the aggregated constraint:

$$\sum_{k=1}^{K} \frac{1}{m_k \overline{s}_k} \sum_{j=1}^{n} p_j x_{kj} \leq K \cdot D. \tag{7}$$

Our algorithm shall make use of the fact that this weaker inequality is sufficient.

Suppose that we replace the constraints (6) by

$$x_{kj} \geq 0, \qquad k = 1, \ldots, K, \; j = 1, \ldots, n. \tag{8}$$

The linear programming problem, minimize D subject to (1), (2), (3), (4), (5), and (8), shall be denoted LP. This linear program can be solved in polynomial time, and its optimal solution $\overline{x}_{kj}$, $k = 1, \ldots, K$, $j = 1, \ldots, n$, and $\overline{C}_j$, $j = 1, \ldots, n$, has an objective function value $\overline{D}$ that is a lower bound on the optimal schedule length, $C^*_{\max}$. This optimal solution to LP can be used to assign the jobs to machine speeds in a good way.

The assignment algorithm is quite simple. For each job j, $j = 1, \ldots, n$, we first compute the (averaged) length of time $\overline{p}_j$ spent processing it by the optimal solution to LP:

$$\overline{p}_j = \sum_{k=1}^{K} (p_j/\overline{s}_k)\overline{x}_{kj}, \qquad j = 1, \ldots, n.$$

Furthermore, we let B_j index the set of (bad) speeds that are too slow for job j:

$$B_j = \{k : p_j/\overline{s}_k > 2\overline{p}_j\}, \qquad j = 1, \ldots, n.$$

For each job j, $j = 1, \ldots, n$, we let $k(j)$ be the index $k \notin B_j$ for which $p_j/(\overline{s}_k m_k)$ is minimized, or equivalently, the one for which $\overline{s}_k m_k$ is largest. For this assignment $k(j)$, $j = 1, \ldots, n$, we shall show that $C + \sum_{k=1}^{K} D_k \leq 2(K+1)\overline{D}$. Since $\overline{D} \leq C^*_{\max}$, by applying the speed-based list scheduling algorithm to this assignment, we have obtained a $2(K+1)$-approximation algorithm.

We first bound the length of any chain under the assignment $k(j)$, $j = 1, \ldots, n$.

Lemma 3.1 *The assignment algorithm computes values $k(j)$, $j = 1, \ldots, n$, such that, for each chain of jobs $\mathcal{C}$,*

$$\sum_{j \in \mathcal{C}} p_j/\overline{s}_{k(j)} \leq 2\overline{D}.$$

Proof: Consider a chain $\mathcal{C}$ consisting of jobs $j_1 \prec j_2 \prec \cdots \prec j_r$. To prove the lemma, we will first show that

$$\sum_{j \in \mathcal{C}} \overline{p}_j \leq \overline{D}. \tag{9}$$

Then, since our algorithm ensures that each job j is processed by a machine $k(j)$ on which it takes at most $2\overline{p}_j$ time units, we obtain the claimed bound.

Since $\overline{x}$ satisfies (3), $\overline{p}_{j_1} \leq \overline{C}_{j_1}$. Furthermore, $\overline{x}$ satisfies (4), and so, since $j_{i-1} \prec j_i$, $i = 2, \ldots, r$, we have that $\overline{p}_{j_i} \leq \overline{C}_{j_i} - \overline{C}_{j_{i-1}}$, $i = 2, \ldots, r$. If we add these constraints, we see that

$$\sum_{j \in \mathcal{C}} \overline{p}_j \leq \overline{C}_{j_r}.$$

Finally, by (5), $\overline{C}_{j_r} \leq \overline{D}$, and hence, (9) holds. ■

We next bound the total load implied by our assignment, $k(j)$, $j = 1, \ldots, n$.

Lemma 3.2 *Let $\widehat{x}_{kj} = 1$ if $k = k(j)$, $j = 1, \ldots, n$, and let $\widehat{x}_{kj} = 0$, otherwise. Then*

$$\sum_{k=1}^{K} \frac{1}{m_k \overline{s}_k} \sum_{j=1}^{n} p_j \widehat{x}_{kj} \leq 2K\overline{D}.$$

Proof: Consider the problem of optimizing over a simplex: minimize $\sum_{j=1}^{N} c_j y_j$ subject to $\sum_{j=1}^{N} y_j = 1$, $y_j \geq 0$, $j = 1, \ldots, N$. In this case, it is easy to see that there exists an optimal solution y^* that is integer: find j^* such that $c_{j^*} = \min_j c_j$, and set $y^*_{j^*} = 1$, and set all other components of y^* to 0.

Similarly, the solution $\widehat{x}$ is an optimal solution to the following linear program:

$$\text{Minimize} \sum_{k=1}^{K} \frac{1}{m_k \overline{s}_k} \sum_{j=1}^{n} p_j x_{kj} \tag{10}$$

subject to

$$\begin{aligned}\sum_{k=1}^{K} x_{kj} &= 1, \qquad j = 1,\ldots,n, &(11)\\ x_{kj} &= 0, \qquad \text{if } k \in B_j,\ j = 1,\ldots,n, &(12)\\ x_{kj} &\geq 0, \qquad k = 1,\ldots,K,\ j = 1,\ldots,n. &(13)\end{aligned}$$

We shall exhibit a feasible solution $\widetilde{x}$ to this linear program for which the objective function value is at most $2K\overline{D}$; hence, the objective function value of $\widehat{x}$ is also at most $2K\overline{D}$, which implies the lemma.

We shall construct the claimed feasible solution by applying the filtering technique of Lin & Vitter (1992) to $\overline{x}$, the optimal solution to LP. Let $\alpha_j := \sum_{k\notin B_j} \overline{x}_{kj}$; note that $\alpha_j > \frac{1}{2}$, since at most half of a weighted average can be more than twice the average. We then set

$$\widetilde{x}_{kj} := \begin{cases} 0 & \text{if } k \in B_j \\ \frac{\overline{x}_{kj}}{\alpha_j} & \text{otherwise} \end{cases}$$

It is straightforward to see that $\widetilde{x}$ is a feasible solution to (11)-(13). Furthermore, $\widetilde{x}_{kj} \leq 2\overline{x}_{kj}$, for each $k = 1,\ldots,K$, $j = 1,\ldots,n$. Hence, from (2), we have that

$$\frac{1}{m_k\overline{s}_k}\sum_{j=1}^{n} p_j\widetilde{x}_{kj} \leq 2\overline{D}, \qquad k = 1,\ldots,K. \tag{14}$$

This implies that $\widetilde{x}$ has objective function value (10) at most $2K\overline{D}$; since $\widehat{x}$ is an optimal solution, it too must have objective function value at most $2K\overline{D}$. ■

By combining Lemmas 3.1 and 3.2, we see that for our assignment $k(j)$, $j = 1,\ldots,n$, $C+\sum_{k=1}^{K} D_k \leq 2(K+1)\overline{D}$. In fact, this argument does not properly balance the contribution of chain and overall machine loads. Instead, we should modify the definition of B_j, by replacing the 2 by $\sqrt{K}+1$; that is,

$$B_j = \{k : p_j/\overline{s}_k > (\sqrt{K}+1)\,\overline{p}_j\}, \qquad j = 1,\ldots,n.$$

We then use this definition of B_j to find an assignment $k(j)$, $j = 1,\ldots,n$, as before: let $k(j)$ be the index $k \notin B_j$ for which $\overline{s}_k\overline{m}_k$ is largest. After modifying the definitions in this way, we then see that a proof exactly analogous to that of Lemma 3.1 yields the following claim.

Lemma 3.3 *The assignment algorithm computes values $k(j)$, $j = 1,\ldots,n$, such that, for each chain of jobs $\mathcal{C}$,*

$$\sum_{j\in\mathcal{C}} p_j/\overline{s}_{k(j)} \leq (\sqrt{K}+1)\overline{D}.$$

■

If one traces through the proof of Lemma 3.2 with this modified definition of B_j, then one finds that α_j is now greater than $\sqrt{K}/(\sqrt{K}+1)$. Consequently, equation (14) becomes

$$\frac{1}{m_k\overline{s}_k}\sum_{j=1}^{n} p_j\widetilde{x}_{kj} \leq \frac{\sqrt{K}+1}{\sqrt{K}}\overline{D}, \qquad k = 1,\ldots,K,$$

and hence we obtain the following analogue of Lemma 3.2.

Lemma 3.4 *Let $\widehat{x}_{kj} = 1$ if $k = k(j)$, $j = 1, \ldots, n$, and let $\widehat{x}_{kj} = 0$, otherwise. Then*

$$\sum_{k=1}^{K} \frac{1}{m_k \overline{s}_k} \sum_{j=1}^{n} p_j \widehat{x}_{kj} \leq (K + \sqrt{K})\overline{D}.$$

■

By combining these lemmas with Theorem 2.1, we have obtained the following result.

Theorem 3.5 *The assignment algorithm combined with the speed-based list scheduling algorithm yields a $(K + 2\sqrt{K} + 1)$-approximation algorithm for $Q|prec|C_{\max}$, where K denotes the number of distinct machine speeds.*

Furthermore, the proof of this theorem actually implies the following somewhat stronger claim.

Corollary 3.6 *The assignment algorithm combined with the speed-based list scheduling algorithm produces a schedule for which $C_{\max}$ is within a factor of $(K + 2\sqrt{K} + 1)$ of $\overline{D}$, the optimal value to the linear program LP.*

Our next goal is to show that we can reduce the general problem to the case in which there are at most $\log m$ distinct speeds. We can assume that the speeds of the m machines s_i, $i = 1, \ldots, m$, actually consist of K distinct speeds $1 = \overline{s}_1 > \overline{s}_2 > \cdots > \overline{s}_K$. (Of course, in the worst case, $K = m$.) We will show that for these machine speeds we can adjust them downwards, so that there are at most $\log m$ remaining distinct speeds, and yet the optimal value of the linear program LP is increased by at most a factor of 4 by this modification. Hence, by Corollary 3.6, we obtain an $O(\log m)$-approximation algorithm for $Q|prec|C_{\max}$. This modification of the speeds consists of two steps, in which we first eliminate all machines whose speed is more than a factor of m slower than the fastest machine, and then simply round down each remaining speed to the nearest power of $(1/2)$. By ensuring that each machine is modified to run no faster than its true speed, we guarantee that any schedule for the modified instance can be interpreted as a schedule of no greater length for the original instance.

Let $k_\circ$ index the smallest speed that is greater than $1/m$; consequently, $\overline{s}_k \leq 1/m$, $k = k_\circ + 1, \ldots, K$. We shall call the machines of speed greater than $1/m$ *fast*, and the remaining machines *slow*. We will show that all work assigned to slow machines can be reassigned to the group of fastest machines. Since there are fewer than m slow machines, the total speed of the slow machines is less than 1, which is equal to s_1. In essence, this means that any machine of speed 1 can do the work of all of the slow machines, while taking at most twice as long. More precisely, for any feasible solution to LP, $\overline{x}_{kj}$, $k = 1, \ldots, K$, $j = 1, \ldots, n$, $\overline{C}_j$, $j = 1, \ldots, n$, and $\overline{D}$, we can construct a feasible solution $\widetilde{x}$ of objective function value at most $2\overline{D}$ that uses only the fast machines. We set $\widetilde{D} = 2\overline{D}$, and for each job $j = 1, \ldots, n$, we set

$$\widetilde{x}_{1j} = \overline{x}_{1j} + \sum_{k_\circ+1}^{K} \overline{x}_{kj},$$

$\widetilde{x}_{kj} = \overline{x}_{kj}$, $k = 2, \ldots, k_\circ$, and $\widetilde{C}_j = \overline{C}_j$. This is a feasible solution to LP for the instance in which only the fast machines remain.

Hence, we may assume that each machine speed $\overline{s}_k \in (1/m, 1]$. For each machine i, round down its speed to the nearest power of $(1/2)$; each machine of speed $s_i \in (2^{-k}, 2^{-k+1}]$ is simulated by a machine of speed 2^{-k}. Another simple calculation shows that the solution $\widetilde{x}_{kj}$, $k = 1, \ldots, k_\circ$,

$j = 1, \ldots, n$, $2\widetilde{C}_j$, $j = 1, \ldots, n$, and $2\widetilde{D} = 4\overline{D}$ is a feasible solution to LP for this rounded data. There are at most $\log m$ distinct speeds after this rounding. Hence, we have obtained a $4(\log m + 2\sqrt{\log m} + 1)$-approximation algorithm for $Q|prec|C_{\max}$.

The construction above to reduce the general case to one in which there are few distinct speeds is not optimized to produce the best leading constant in the resulting performance guarantee. For example, we can round all speeds less than $1/(\beta m)$ down to 0, and round all speeds in the interval $(\alpha^{-k}, \alpha^{-k+1}]$ down to α^{-k}, for any choice of $\alpha > 1$ and $\beta \geq 1$. The same calculations imply that there are $\log_\alpha(\beta m)$ distinct speeds resulting, and the rounding increases the optimal value of the linear program LP by a factor that is at most $\alpha(1 + 1/\beta)$. If we set $\alpha = e$ and $\beta = \log m$ (and we should emphasize at this point that we have been using $\log m$ to denote $\log_2 m$), then the performance guarantee becomes $1.89 \log m + \Theta(\sqrt{\log m})$. Thus, we have established the following theorem.

Theorem 3.7 *If K is the number of different machine speeds, there is a $\min\{K+2\sqrt{K}+1, 1.89\log m + \Theta(\sqrt{\log m})\}$-approximation algorithm for $Q|prec|C_{\max}$.*

In fact, we have shown that given a feasible solution to LP with objective value D, there is a polynomial-time algorithm that finds a schedule of length at most $\min\{K + 2\sqrt{K} + 1, 1.89 \log m + \Theta(\sqrt{\log m})\}D$. We will use this fact in the next section.

Now consider the generalization of this scheduling problem in which each job j, $j = 1, \ldots, n$, has a release date r_j before which it may not begin processing. As mentioned earlier, we can apply a general result of Shmoys, Wein, & Williamson (1995) that, given our approximation algorithm for $Q|prec|C_{\max}$, produces an algorithm for $Q|r_j, prec|C_{\max}$, with twice the performance guarantee. One must be a bit careful in applying this technique, since it is necessary that $j \prec j'$ implies that $r_j \leq r_{j'}$, but this is easy to ensure without loss of generality.

Corollary 3.8 *For $Q|r_j, prec|C_{\max}$, there is a $2 \cdot \min\{K + 2\sqrt{K} + 1,\ 1.89\log m\ + \Theta(\sqrt{\log m})\}$-approximation algorithm, where K is the number of distinct machine speeds.*

Next, notice that our linear programming relaxation is also a relaxation for the variant of the problem in which preemption is allowed. As a consequence, we obtain the following corollary, which improves upon the $O(\sqrt{m})$-approximation algorithm of Jaffe (1980).

Corollary 3.9 *For $Q|prec, pmtn|C_{\max}$, there is a $\min\{K + 2\sqrt{K} + 1,\ 1.89\log m\ + \Theta(\sqrt{\log m})\}$-approximation algorithm, where K is the number of distinct machine speeds.*

Furthermore, since our algorithm produces schedules that do not preempt any jobs, we have established the following interesting relationship.

Corollary 3.10 *For any instance of $Q|prec|C_{\max}$, the ratio between its nonpreemptive optimal value and its preemptive optimal value is $O(\log m)$.*

We will show next that we can not obtain a constant performance guarantee by relying only on LP for the lower bound used in the proof of theorem (3.7). Perhaps more surprisingly, even the integer programming version, that is, where the constraints (8) are replaced by (6), is itself a substantial relaxation of the original problem $Q|prec|C_{\max}$. The next theorem shows that, in comparing the optimal schedule length $C^*_{\max}$ to the optimal value D^* of this integer program, our analysis is nearly tight.

Theorem 3.11 *There are instances of $Q|prec|C_{\max}$ for which $C^*_{\max} = \Omega(\log m / \log\log m)D^*$.*

Proof: Consider the instances of $Q|prec|C_{\max}$ defined in the following way. Let $\overline{m} = 2^{\kappa}$, $\gamma = \log\kappa$ and $\eta = \kappa/\gamma$. We shall set $D = \overline{m}\kappa^{\eta+1}$, but this merely scales the time units in this construction. There are $m_1 = 2\kappa$ machines of speed $s_1 = \overline{m}/\kappa$, $m_2 = (2\kappa)^2$ machines of speed $s_2 = \overline{m}/\kappa^2$, ..., and $m_\eta = (2\kappa)^\eta$ machines of speed $s_\eta = \overline{m}/\kappa^\eta$. Thus, the number of machines m satisfies $\overline{m} \leq m \leq 2\overline{m}^2$, which implies that $\kappa = \Theta(\log m)$, $\gamma = \Theta(\log\log m)$ and $\eta = \Theta(\log m/\log\log m)$. Notice also that $m_k s_k = 2^k\overline{m}$, $k = 1, \ldots, \kappa$.

There are η groups of jobs J_i, $i = 1, \ldots, \eta$. The first group contains $2\kappa^2$ jobs of processing requirement $\overline{m}D/\kappa^2$, the second contains $2^2\kappa^3$ jobs of processing requirement $\overline{m}D/\kappa^3$, and so forth, through the last group, that contains $2^\eta\kappa^{\eta+1}$ jobs of processing requirement $\overline{m}D/\kappa^{\eta+1}$. For each job $j \in J_i$ and each job $j' \in J_{i+1}$, $i = 1, \ldots, \eta - 1$, we have that $j \prec j'$.

If each job in the group J_i is exactly assigned to be processed by a machine of speed s_i, for each $i = 1, \ldots, \eta$, we obtain a feasible solution to the integer program with objective value D; hence $D^* = O(D)$. We show next that each J_i, $i = 1, \ldots, \eta - 1$, needs $\Omega(D)$ time to complete; thus, given the structure of the precedence graph, each schedule completes at time $\Omega(D\eta) = \Omega(D\log m/\log\log m)$, which proves the theorem.

Consider any schedule for J_i, for some $i = 1, \ldots, \eta - 1$. Since we never use more machines than jobs, we can assume that we are only using the fastest $|J_i| = 2^i\kappa^{i+1}$ machines. Since the number of machines of speed s_{i+1} is greater than $|J_i|$, we will only be using machines of speeds $s_1, \ldots, s_{i+1}$. Hence the total speed of all machines used to process J_i is at most

$$\sum_{k=1}^{i+1} m_k s_k = \sum_{k=1}^{i+1} 2^k\overline{m} \leq 2^{i+2}\overline{m}.$$

Since the total processing requirement is

$$2^i\kappa^{i+1}\frac{\overline{m}D}{\kappa^{i+1}} = 2^i\overline{m}D,$$

any (preemptive or nonpreemptive) schedule for J_i is of length at least

$$\frac{2^i\overline{m}D}{2^{i+2}\overline{m}} = \frac{D}{4}.$$

■

In considering the gap between the $O(\log m)$ performance guarantee and the $\Omega(\log m/\log\log m)$ lower bound in Theorem 3.11, we suspect that it might be possible to improve our algorithm. In particular, we think that it might be possible to round the data for any instance to one with K distinct speeds while increasing the optimal value of the linear program LP by a factor of at most ρ, where $\rho K = O(\log m/\log\log m)$. Of course, one advantage of our analysis is that whenever one can round the input so that ρK is small, there is an improved performance guarantee implied as well.

4 An extension for $Q|r_j, prec|\sum w_jC_j$

In this section we give an $O(\log m)$-approximation algorithm for $Q|r_j, prec|\sum w_jC_j$, which is based on combining our algorithm for $Q|prec|C_{\max}$ with a technique for the special case in which the machines are identical, $P|r_j, prec|\sum w_jC_j$, that was introduced by Hall, Schulz, Shmoys & Wein (1996).

We shall start by describing the key points of the result of Hall, Schulz, Shmoys, & Wein for identical machines. First, they subdivide the time horizon into the following intervals of potential job completion times: $[1,2],(2,4],(4,8],\ldots$. Suppose that we know just the interval that contains C_j^*, for each $j=1,\ldots,n$, where C_j^* denotes the completion time of job j in some optimal schedule: this rough information can be used to derive a schedule in which each job j completes by time $8C_j^*$, $j=1,\ldots,n$. We shall construct a fragment of the schedule for the jobs that complete in the interval $(2^{\ell-1},2^\ell]$ by the standard list scheduling algorithm (ignoring release dates). Graham's analysis ensures that the length of this fragment is at most $2(2^\ell)$. Each job j for which $C_j^* \leq 2^\ell$ must also have $r_j \leq 2^\ell$. Hence, if we actually run the fragment for the ℓth interval during the time period $(4\cdot 2^{\ell-1}, 4\cdot 2^\ell]$, we have obtained a feasible schedule. Furthermore, if $C_j^* \in (2^{\ell-1},2^\ell]$, then it completes in this schedule by $8(2^{\ell-1})$, which implies that the algorithm is an 8-approximation algorithm. Of course, we do not know the interval in an optimal schedule in which each job finishes; Hall, Schulz, Shmoys, & Wein show that a linear programming relaxation can be used to provide sufficient estimates, and in fact give a 7-approximation algorithm for $P|r_j,prec|\sum w_jC_j$.

We shall extend their technique to uniformly related machines in a natural way. We shall use a linear relaxation in order to estimate the interval in which each job completes. This assigns a set of jobs to each interval, which will be scheduled using our algorithm of Section 3. This fragment of the schedule will then be run in a shifted time period, which is sufficiently long, and starts sufficiently late so as to satisfy all release-date constraints. As in the result of Hall, Schulz, Shmoys, & Wein (1996), we shall use an interval-indexed linear programming relaxation in order to estimate the interval in which each job completes. Let $\overline{s}_1 > \cdots > \overline{s}_K$ denote the distinct speeds in our input; we shall assume without loss of generality that $p_j/\overline{s}_k \geq 1$, for each $k=1,\ldots,K$, $j=1,\ldots,n$. Let $\tau_\ell = 2^\ell$, $\ell=0,\ldots,L$, where $L=\log(\max_j r_j + \sum_j p_j/\overline{s}_K)$. We shall rely on 0-1 decision variables $x_{kj\ell}$, which indicate whether or not job j completes on a machine of speed $\overline{s}_k$ in the ℓth interval (from $2^{\ell-1}$ to 2^ℓ). Since we will bound the length of each fragment by demonstrating a feasible solution to the corresponding instance of LP, we shall also introduce the variables C_j, $j=1,\ldots,n$, which will be used to bound the length of the chains.

Next we describe our linear programming relaxation. Since C_j can be thought of as being the finishing time of job j, the objective is

$$\text{Minimize} \quad \sum_{j=1}^{n} w_j C_j \tag{15}$$

All the jobs must be assigned to run at some machine speed:

$$\sum_{k=1}^{K}\sum_{\ell=1}^{L} x_{kj\ell} = 1, \quad j=1,\ldots,n. \tag{16}$$

The capacity constraints up to time τ_ℓ can be written as follows:

$$\frac{1}{\overline{s}_k m_k}\sum_{j=1}^{n} p_j \sum_{t=1}^{\ell} x_{kjt} \leq \tau_\ell,\ k\text{=1,...,}K,\ \ell\text{=1,...,}L. \tag{17}$$

The time required to execute a job cannot exceed the period of time between its release and its completion:

$$\sum_{k=1}^{K}\frac{p_j}{\overline{s}_k}\sum_{\ell=1}^{L} x_{kj\ell} \leq C_j - r_j, \quad j=1,\ldots,n. \tag{18}$$

As in LP, for precedence constraints we require

$$\sum_{k=1}^{K}\frac{p_j}{\overline{s}_k}\sum_{\ell=1}^{L} x_{kj\ell} \leq C_j - C_{j'}, \quad \text{if } j' \prec j. \tag{19}$$

In addition, if $j' \prec j$, the interval in which j completes cannot precede the one in which j' completes

$$\sum_{t=1}^{\ell}\sum_{k=1}^{K} x_{kjt} \leq \sum_{t=1}^{\ell}\sum_{k=1}^{K} x_{kj't}, \text{ if } j' \prec j, \ \ell\text{=1,...,}L. \tag{20}$$

Since the finishing time of job j is certainly greater than the left endpoint of the interval in which it completes

$$\sum_{\ell=1}^{L} \tau_{\ell-1} \sum_{k=1}^{K} x_{kj\ell} \leq C_j, \ \ j = 1, \ldots, n. \tag{21}$$

And finally, we impose

$$x_{kj\ell} \geq 0, \ \ k\text{=1,...,}K, \ j\text{=1,...,}n \ \ell = 1\text{,...,}L. \tag{22}$$

We first solve the linear program (15)-(22); let $\overline{x}_{kj}$, $k = 1, \ldots, K$, $j = 1, \ldots, n$, and $\overline{C}_j$, $j = 1, \ldots, n$ denote an optimal solution. We shall assign each job j to an interval in the following way, which builds on the half-interval technique introduced by Hall, Shmoys, & Wein (1996). First, for $j = 1, \ldots, n$, let $\ell_1(j)$ denote the minimum value of ℓ for which

$$\sum_{t=1}^{\ell}\sum_{k=1}^{K} \overline{x}_{kjt} \geq 1/2.$$

Next, let $\ell_2(j)$ denote the minimum value of ℓ for which

$$\overline{C}_j \leq 2^{\ell}.$$

We set $\ell(j) = \max\{\ell_1(j), \ell_2(j)\}$ and $J_\ell = \{j : \ell(j) = \ell\}$, $\ell = 1, \ldots, L$. We shall construct a schedule for each subset J_ℓ separately, by applying the algorithm from the previous section. In order to bound the length of the fragment of the schedule for J_ℓ, we will show that the optimal solution $\overline{x}$ and $\overline{C}$ can be used to define a feasible solution to the instance of LP for this subset of jobs. Note that the constraints (19) & (20) ensure that if we concatenate the fragments for $J_1, J_2, \ldots, J_L$, and each fragment satisfies the precedence constraints among the jobs scheduled within it, then the resulting schedule satisfies all precedence constraints.

For each job j, $j = 1, \ldots, n$, let α_j denote the total fraction of job j that is completed in the fractional solution $\overline{x}$, over all speeds, in the first $\ell(j)$ intervals. By our definition of $\ell(j)$, $\alpha_j \geq 1/2$, $j = 1, \ldots, n$. We set

$$\widetilde{x}_{kj} = \sum_{\ell=1}^{\ell(j)} \overline{x}_{kj\ell}/\alpha_j, \qquad k = 1, \ldots, K, \ j = 1, \ldots, n.$$

Fix some $\ell = 1, \ldots, L$, and consider the jobs in J_ℓ. If we set $\widetilde{C}_j = 2\overline{C}_j$, $j \in J_\ell$, and $\widetilde{D} = 2^{\ell+1}$, then we shall argue that $\widetilde{x}$, $\widetilde{C}$, and $\widetilde{D}$ constitute a feasible solution for LP for this subset of jobs.

Clearly, $\widetilde{x}$ satisfies the assignment constraints (1). Since each $\alpha_j \geq 1/2$, and $\overline{x}$ satisfies (18), we have that $\widetilde{x}$ and $\widetilde{C}$ satisfy (3); similarly, the constraints (19) ensure that (4) are satisfied and the constraints (17) ensure that (2) are satisfied. Finally, for each job $j \in J_\ell$, we have that $\overline{C}_j \leq 2^{\ell_2(j)} \leq 2^{\ell(j)} = 2^{\ell}$, and hence $\widetilde{C}_j \leq \widetilde{D}$. Theorem 3.7 implies that we can construct a schedule fragment for J_ℓ of length $\overline{K}2^{\ell+1}$, where $\overline{K}$ is the performance guarantee proved for our approximation algorithm for the $C_{\max}$ objective. This fragment shall be run from time $\overline{K}(1 + 2 + \cdots + 2^{\ell})$ to $\overline{K}(1 + 2 + \cdots + 2^{\ell} + 2^{\ell+1})$. Since $\overline{K} \geq 1$ and $r_j \leq \overline{C}_j \leq 2^{\ell}$ for each job $j \in J_\ell$, we are assured that the resulting schedule satisfies the release-date constraints.

It remains to show that we have scheduled each job j to complete at a time not much greater than $\overline{C}_j$: we shall show that $2^{\ell(j)} \leq 4\overline{C}_j$, $j = 1, \ldots, n$. Since each job $j \in J_\ell$ completes by time $4\overline{K}2^{\ell(j)}$, this will imply that we have derived a $16\overline{K}$-approximation algorithm. We consider two cases. If $\ell(j) = \ell_2(j)$, then $\overline{C}_j \geq 2^{\ell(j)-1}$, and hence $2^{\ell(j)} \leq 2\overline{C}_j$. In the remaining case, $\ell(j) = \ell_1(j) > \ell_2(j)$. Then,

$$\begin{aligned}
\tau_{\ell_1(j)-1}/2 &\leq \tau_{\ell_1(j)-1}(\sum_{k=1}^{K}\sum_{\ell=\ell_1(j)}^{L}\overline{x}_{kj\ell}) \\
&\leq \sum_{k=1}^{K}\sum_{\ell=\ell_1(j)}^{L}\tau_{\ell-1}\overline{x}_{kj\ell} \\
&\leq \sum_{k=1}^{K}\sum_{\ell=1}^{L}\tau_{\ell-1}\overline{x}_{kj\ell} \\
&\leq \overline{C}_j.
\end{aligned}$$

Hence, $2^{\ell(j)} \leq 4\overline{C}_j$, and we have proved the following theorem.

Theorem 4.1 *There is a polynomial-time algorithm for* $Q|prec, r_j|\sum w_jC_j$ *that finds a schedule of objective function value within a factor of* $16\min\{K + 2\sqrt{K} + 1, 1.89\log m + \Theta(\sqrt{\log m})\}$ *of the optimum, where* K *is the number of distinct speeds.*

Our linear programming relaxation is also a relaxation for the variant of the problem in which preemption is allowed. Consequently, we obtain the following corollaries.

Corollary 4.2 *For* $Q|prec, r_j, pmtn|\sum w_jC_j$, *there is a* $16\min\{K+2\sqrt{K}+1, 1.89\log m+\Theta(\sqrt{\log m})\}$-*approximation algorithm, where* K *is the number of distinct speeds,*

Corollary 4.3 *For any given instance of* $Q|prec, r_j|\sum w_jC_j$, *the ratio between its nonpreemptive optimal value and its corresponding preemptive optimal value is* $O(\log m)$.

Acknowledgments We are grateful to Joel Wein for several helpful discussions.

References

Chakrabarti, S., C. Phillips, A. S. Schulz, D. B. Shmoys, C. Stein, J. Wein (1996). Improved scheduling algorithms for minsum criteria. In F. Meyer auf der Heide and B. Monien, eds. *Automata, Languages, and Processing, Proceedings of the 23rd International Colloquium ICALP '96, Lecture Notes in Computer Science 1099*, Springer, Berlin, 646–657.

Gonzalez, T., O.H. Ibarra, and S. Sahni (1977). Bounds for LPT schedules on uniform processors. *SIAM J. Comput.* **6** 155–166.

Graham, R.L. (1966). Bounds for certain multiprocessing anomalies. *Bell System Tech. J.* **45** 1563–1581.

Graham, R.L., E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** 287–326.

Hall, L.A., A.S. Schulz, D.B. Shmoys, J. Wein (1996). *Scheduling to minimize average completion time: off-line and on-line approximation algorithms.* Technical Report 1159, School of Operations Research and Industrial Engineering, Cornell University, Ithaca, NY, USA.

Hall, L. A., D. B. Shmoys, J. Wein (1996). Scheduling to minimize average completion time: off-line and on-line algorithms. *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, 142–151.

Hochbaum, D.S., D.B. Shmoys (1988). A polynomial approximation scheme for scheduling on uniform processors: using the dual approximation approach. *SIAM J. Comput.* **17** 539–551.

Jaffe, J. (1980). Efficient scheduling of tasks without full use of processor resources. *Theoret. Comput. Sci.* **12** 1–17.

Lenstra, J.K., A.H.G. Rinnooy Kan (1978). Complexity of scheduling under precedence constraints. *Oper. Res.* **26** 22–35.

Lin, J.H., J.S. Vitter (1992). ϵ-approximation with minimum packing constraint violation. *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, 771–782.

Liu, J.W.S., C.L. Liu (1974). Bounds on scheduling algorithms for heterogeneous computing systems. in: J.L. Rosenfeld (ed.), *Information Processing 74*, North-Holland, 349–353.

Schulz, A.S. (1996). *Scheduling and Polytopes.* PhD thesis, Technische Universität Berlin, Berlin, Germany.

Shmoys, D.B., É. Tardos (1993). An approximation algorithm for the generalized assignment problem. *Math. Programming* **62** 461–474.

Shmoys, D.B., J. Wein, D.P. Williamson (1995). Scheduling parallel machines on-line. *SIAM J. Comput.* **24** 1313–1331.